Christopher Fairchild

Advanced Data Management – D191

**Section A: Business Purpose and Report**

Successful businesses are those which effectively respond to changes in trends in their target market. In the case of a DVD-rental business, one of the trends which should be closely monitored is that of film popularity--adjusting DVD inventory to compensate for shifts in overall film popularity will allow the business to remain competitive and profitable. I propose the creation of a report which will show which films are most popular based on the total number of times each film has been rented—first broken down by inventory item, and then by the film itself. This report will also address a problem uniquely associated with rental businesses—that is, the frequency with which inventory items are damaged. By monitoring the number of times each inventory item has been rented, business owners will be able to better predict when items will need to be cleaned, repaired, or replaced.

The report would be built using rental data, inventory and film data, as well as category data. The rental data will be used to determine how many times each inventory item has been rented, thus creating the initial ranking of inventory items. The inventory and film data will be used to associate each inventory item with the film it contains; the business carries multiple copies (i.e., inventory items) of some films, so this data is necessary to accurately rank each film. Finally, the category data will be used to determine the category for each film.

The detailed section of the report will display data from the rental, inventory, film, film\_category and category tables. The number of columns will be reduced in the summary section of the report, and thus will only display data gathered from the rental, film, and category tables.

The detailed section will display the following information across six columns:

1. Rental count (represented by an integer), which will use a count of the rental\_idfield from the rental table;
2. an inventory ID (represented by an integer) for each item, using the inventory\_idfield from the inventory table;
3. a film id (represented by an integer) for each film in the store’s stock, using the film\_idfield from the film table;
4. a title(represented as a string) for each film from the film table;
5. a category ID (represented by an integer) for each film, using the category\_idfield from the film\_category table; and
6. a name(represented as a string) for each category from the category table.

The summary section will be composed of the following three columns, using the same data types described in the detailed section:

1. Rental count, which will use a count of the rental\_idfield from the rental table;
2. a titlefor each film from the film table, and
3. a namefor each category from the category table.

To obtain the initial popularity ranking of each inventory item (inventory\_id), each item’s rental count will need to be calculated. This will be achieved by using the COUNT function on the rental\_id column in the rental table and grouping the results by inventory\_id. This will provide the user with a more convenient way of viewing the rental count for each inventory item. Transforming the data in this way is non-negotiable; without obtaining the total rental counts, the report user would have to sort through over 16,000 rows of data to determine which inventory items have the highest rental count.

An additional transformation will be carried out using the “UPDATE\_NULL\_ID” function. There are several films represented in the DVD rental database which are not currently in stock; this function searches for these films and assigns a value of 0 to the inventory\_id column. Without running this function, any films not currently in stock will have an inventory\_id of “[null],” which may be confusing to users who are less familiar with the way SQL manages data. A value of 0 is a much more beginner-friendly visual cue for a lack of inventory items for that specific film.

The extra information provided in the detailed section of the report will allow the report user to determine if there are any specific DVDs that are being rented at a significantly higher frequency, which would provide insight into which inventory items should be inspected for excessive wear or damage. The summary section will essentially provide the same benefit as the detailed section (i.e., ranking popularity based on rental count), but will focus on each individual film rather than each inventory item. This will help the report user monitor the popularity of a given film over time--if a film begins to be rented more frequently, the report user may choose to purchase more copies of that film in order to keep up with the spike in demand.

To remain competitive, the business owners should refresh this report once per month at a minimum. By analyzing rental data on a regular basis, they will be able to quickly adjust their inventory to meet changing demands in the rental market, which will likely be influenced by each new release, as well as reviews and trends on social media. Additionally, the business owners will be able to determine the frequency with which inventory items should be inspected for wear and tear, allowing them to clean, repair, or replace an item before it becomes too damaged to be rented.

**Section B: SQL Table Creation Code**

Detailed report table:

CREATE TABLE IF NOT EXISTS detailed\_report(  
times\_rented INT,  
inventory\_id INT,  
film\_id INT,  
film\_title VARCHAR(255),  
category\_id INT,  
category\_name VARCHAR(25)  
);

Summary report table:

CREATE TABLE IF NOT EXISTS summary\_report(  
rental\_count INT,  
film\_title VARCHAR(255),  
category\_name VARCHAR(25)  
);

**Section C: SQL Data Extraction and Detailed Report Data Population**

NOTE: Populating the detailed report requires that the UPDATE\_NULL\_ID() function be created. The function definition can be found in Section D.

INSERT INTO detailed\_report (times\_rented, inventory\_id, film\_id, film\_title, category\_id, category\_name)SELECT COUNT(rental.rental\_id) AS times\_rented, (SELECT UPDATE\_NULL\_ID(inventory.inventory\_id)),film.film\_id, film.title, film\_category.category\_id, category.nameFROM rentalFULL JOIN inventory ON rental.inventory\_id = inventory.inventory\_idRIGHT JOIN film ON film.film\_id = inventory.film\_idLEFT JOIN film\_category ON film\_category.film\_id = film.film\_id LEFT JOIN category ON film\_category.category\_id = category.category\_idGROUP BY film.film\_id, category.name, inventory.inventory\_id, film\_category.category\_idORDER BY times\_rented;

**Data Accuracy Verification**

1. To verify that the detailed report table accounts for all items in the inventory table:

SELECT COUNT(detailed\_report.inventory\_id) AS report\_count,

COUNT(inventory.inventory\_id) AS inventory\_count

FROM detailed\_report

FULL JOIN inventory ON detailed\_report.inventory\_id = inventory.inventory\_id

WHERE detailed\_report.inventory\_id != 0;

1. To verify the number of rentals for a given inventory\_id:

SELECT COUNT(rental\_id)

FROM rental

WHERE inventory\_id =766

*(replace 766 with the inventory\_id you wish to verify)*

1. To verify the film\_id associated with a given inventory\_id:

SELECT film\_id

FROM inventory

WHERE inventory\_id = 766

*(replace 766 with the inventory\_id you wish to verify)*

1. To verify the title associated with a given film\_id:

SELECT title

FROM film

WHERE film\_id = 167

*(replace 167 with the film\_id you wish to verify, which can be retrieved using the previous query)*

1. To verify the category\_id associated with a given film\_id:

SELECT category\_id

FROM film\_category

WHERE film\_id = 167

*(replace 167 with the film\_id you wish to verify)*

1. To verify the name associated with a given category\_id

SELECT “name”

FROM category

WHERE category\_id = 16

*(replace 16 with the category\_id you wish to verify, which can be retrieved using the previous query)*

**Section D: Data Transformation**

The data transformation described in Section A (that is, the generation of a consolidated rental count for each inventory item) is demonstrated using the following code blocks in Section C:

SELECT COUNT(rental.rental\_id) AS times\_rented, inventory.inventory\_id,film.film\_id, film.title, film\_category.category\_id, category.name

GROUP BY film.film\_id, category.name, inventory.inventory\_id, film\_category.category\_id

In order to highlight the critical components used in the transformation, a similar result can be seen using the following query:

SELECT COUNT(rental\_id) AS times\_rented, inventory\_id  
FROM inventory  
GROUP BY inventory\_id;

The custom data transformation of null values described in Section A is accomplished using the following function:

CREATE FUNCTION UPDATE\_NULL\_ID(target\_id INT) RETURNS INT AS $UPDATE\_NULL\_ID$

BEGIN

IF @target\_id IS NULL THEN

target\_id = 0;

END IF;

RETURN @target\_id;

END

$UPDATE\_NULL\_ID$ LANGUAGE plpgsql;

**Section E: Summary Report Update Trigger**

The following function contains the functionality to populate and update data within the summary report table. As such, it must be created first in order for the trigger to function.

CREATE FUNCTION POPULATE\_SUMMARY\_REPORT() RETURNS TRIGGER AS $POPULATE\_SUMMARY\_REPORT$

BEGIN

DELETE FROM summary\_report;

INSERT INTO summary\_report(rental\_count, film\_title, category\_name)

SELECT SUM(times\_rented) AS rental\_count, film\_title, category\_name

FROM detailed\_report

GROUP BY film\_title, category\_name

ORDER BY rental\_count DESC;

RETURN NEW;

END;

$POPULATE\_SUMMARY\_REPORT$ LANGUAGE plpgsql;

The following trigger will update the summary report table using the above function after each UPDATE is performed on the detailed report table.

CREATE TRIGGER UPDATE\_SUMMARY

AFTER UPDATE ON detailed\_report

FOR EACH STATEMENT

EXECUTE FUNCTION POPULATE\_SUMMARY\_REPORT();

**Section F: Stored Procedure for Data Refresh**

The following stored procedure is used to refresh the data in both the detailed and summary report tables. When this procedure is called, all data within the detailed and summary tables will be deleted and replaced with the data contained in the dvdrental database.

CREATE PROCEDURE REFRESH\_TABLES() AS $REFRESH\_TABLES$

BEGIN

DELETE FROM detailed\_report;

INSERT INTO detailed\_report (times\_rented, inventory\_id, film\_id, film\_title, category\_id, category\_name)

SELECT COUNT(rental.rental\_id) AS times\_rented, inventory.inventory\_id,film.film\_id, film.title, film\_category.category\_id, [category.name](http://category.name/)

FROM rental

FULL JOIN inventory ON rental.inventory\_id = inventory.inventory\_id

RIGHT JOIN film ON film.film\_id = inventory.film\_id

LEFT JOIN film\_category ON film\_category.film\_id = film.film\_id

LEFT JOIN category ON film\_category.category\_id = category.category\_id

GROUP BY film.film\_id, [category.name](http://category.name/), inventory.inventory\_id, film\_category.category\_id

ORDER BY times\_rented DESC;

DELETE FROM summary\_report;

INSERT INTO summary\_report(rental\_count, film\_title, category\_name)

SELECT SUM(times\_rented) AS rental\_count, film\_title, category\_name

FROM detailed\_report

GROUP BY film\_title, category\_name

ORDER BY rental\_count DESC;

END;

$REFRESH\_TABLES$ LANGUAGE plpgsql;

This refresh procedure should, at a minimum, be used monthly. Regularly refreshing the data within the detailed and summary reports will ensure that the report user is basing their business decisions on the most up-to-date data. Refreshing the data less frequently may negatively affect the user’s ability to interpret trends, thus causing them to miss opportunities to adjust to changes in demand.

To automate the above-described monthly refresh, I suggest the rental business owners consider installing the “pg\_cron” extension in their company database (CitusData, 2022). Once the extension has been installed and configured in the database, scheduling the refresh can be accomplished using the following SQL query, which will run the REFRESH\_TABLES stored procedure on the first day of every month at 12:00 AM GMT:

SELECT cron.schedule(‘0 0 1 \* \*', ‘CALL REFRESH\_TABLES();’);

**References:**

CitusData. (2022, July 15). pg\_cron repository: README.md. *Github*.

<https://github.com/citusdata/pg_cron/blob/main/README.md>